Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR-TP-99272

Quality first

Measuring a safety-critical embedded software development process

E. Kesseler

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-99272

Quality first

Measuring a safety-critical embedded software development process

E. Kesseler

This report is based on a presentation held at the VTT Symposium, Oulu, Finland, June 22-24, 1998.

Division:	Information and Communication Technology
Issued:	August 1999
Classification of title:	unclassified

Abstract

Software which is embedded in aircraft to which people entrust their lives becomes safety-critical and consequently must be of the highest quality. Failures of such software must be virtually non-existent. Due to the high costs of aircraft, hypothetical software failures would also incur major financial losses. To guarantee that the safety of aircraft is an uncompromisable requirement, an independent government agency certifies aircraft as fit-for-use.

The experience with a software development process model accommodating both safety-critical requirements as well as commercial requirements is described. The findings are based on process and product metrics. The first two versions of the software product have successfully passed the certification and are currently being flown in numerous aircraft. In fact the software product is so successful that it will be adapted to other aircraft models.

Measuring the requirements evolution contributed to the change from a waterfall based software development process to a spiral software development process. Design stability is reflected in the module evolution but needs to be complemented by other information. Requirements evolution and its implementation status combined with design stability help in the trade-off between additional deliveries, their functions and their release dates.

Abbreviations

CASE	Computer Aided Software Engineering
COTS	Commercial-Of-The-Shelf
EFIS	Electronic Flight Instrument System
FAR	Federal Airworthiness Requirement
HMI	Human Machine Interface
IMC	Instrument Meteorological Conditions
JAR	Joint Aviation Requirement
NTSB	(US) National Transport Safety Board
MC/DC	modified condition/ decision coverage
SA / RT	Structured Analysis / Real Time extensions
SD	Structured Design
TCD	Test Case Definition
VMC	Visual Meteorological Conditions

Contents

1 Introduction

Software which is embedded in aircraft to which people entrust their lifes becomes safety-critical and consequently must be of the highest standards. Failures of such software must be so rare as virtually non-existing during the lifetime of all aircraft concerned. Due to the high costs of aircraft, hypothetical software failures would also incur major financial losses, a further drive to require the highest quality. It is clear that in aircraft, safety is an uncompromisable requirement.

To guarantee the safety of aircraft, an independent government agency certifies aircraft as fit-for-use. Only after this certification the aircraft may be used commercially. To guarantee a world-wide equal level of safety, software airworthiness requirements are stated in one document, [DO-178B]. This document contains information for both the certification authorities and the developers.

A software development process based on the waterfall model is a well-proven way to produce safety-critical software. [DEKK, KESS] provides an example where an ESA-PSS05 compliant process is used. For complex technical systems like aircraft, the commercially determined time-to-market mandates co-development of the various subsystems. Co-development will inevitably result in requirements evolution. Even more so if complicated Human Machine Interfaces are involved. The waterfall model is not intended to cope with such requirements evolution.

The experience with a DO-178B compliant software development process which accommodates a commercial time-to-market is described. The findings are based on process and product metrics. The first two product versions have successfully passed the certification and are currently being flown in numerous aircraft. In fact the software product is so successful that it will be adapted to other aircraft models.

The sequel starts with a short description of the application. Subsequently some information about the air transport safety requirements is provided, together with its influence on the software development process to be applied. The experience gained during the production of the embedded application is described, supported by metrics. The findings are summarised in the conclusions.

2 Application description

To fly aircraft under all (adverse) conditions, pilots must fully rely on the data presented to them, and on the correct and timely forwarding of their commands to the relevant aircraft subsystems. The embedded avionics application discussed combines, controls, processes and forwards the data between the subsystems and the flight deck. The flight deck may contain conventional mechanical displays or a modern Electronic Flight Instrument System (EFIS) or even a mix of these. The application generates all information for the flight deck as well as processes all pilot inputs. This renders the application vulnerable to changes in the aircraft's Human Machine Interfaces.

The embedded application is designed to operate in both Visual Meteorological Conditions (VMC) and Instrument Meteorological Conditions (IMC). In the former conditions, the pilot can obtain part of the necessary flight information from visual cues from outside the cockpit. These conditions limit the aircraft operations to good weather operations. The latter conditions allow all-weather operations of the aircraft. Under these conditions the displays of the flight deck are needed by the pilot to fly. This renders the correct functioning of the displays safety-critical. A number of equipment items needs to be duplicated to achieve the required low failure probability.

During normal operations the embedded application processes about 100 different flight parameters, originating from 10 different sensors, some of which are duplicated. Two processors are used in each of the duplicated hardware units. The delay times within the entire embedded application should be guaranteed to be less then 30 milliseconds with a cycle time of 25 milliseconds for the main processor. During the operational life of the embedded application many extensions are expected, so 50% spare processor time shall be allowed for. The I/O processor has a cycle time of 360 microseconds.

The influence of safety on the embedded application's functions will be illustrated for data input. Depending on the criticality of the flight parameter, the software validates it in up to four complementary ways:

- coherency test: a check on correct length and parity of the data;
- reception test: a check on the timely arrival of the data;
- sensor discrepancy test: a comparison between the two data values produced by the two independent redundant sensors; and
- module discrepancy test: a comparison between the two parameter values produced by the same sensor; one value directly read by the system from the sensor, and one obtained from the redundant system via a cross-talk bus.

[Kess, Slui] contains more information on the application.

3 Air transport safety requirements

3.1 Applicable software safety document

For safety-critical software in airborne equipment [DO-178B] has been developed. This document provides guidance for both the software developers and the certification authorities. In civil aviation an independent governmental institution, the certification authority, performs the ultimate system acceptance by certifying the entire aircraft. Only then the constituent software is airworthy and ready for commercial use. [DO-178B] provides a world-wide "level playing field" for the competing industries as well as a world-wide protection of the air traveller, which are important due to the international character of the industry. In NLR's case the certification authority concerned delegated some of its technical activities to a specialised company.

Certifying the entire aircraft implies that when an aircraft operator wants an aircraft with substantial modifications, the aircraft including its embedded software has to be re-certified. Substantial modifications are, for example, modifications which can not be accommodated by changing the certified configuration files.

[DO-178B] was the first widely used document to address safety-critical software. Based on amongst others the experience gained with this document, currently other more general-purpose standards are available, like [ISO/DIS 15026] and [IEC 61508]. [SAE ARP 4754] addresses the certification considerations for highly-integrated or complex aircraft systems. [SAE ARP 4754] is complementary to [DO-178B] and applicable hardware specific standards.

3.2 Safety classification

Based on the impact of the system (i.e. aircraft) failure the software failure can contribute to, the software is classified into 5 levels. The failure probability in flight hours (i.e. actual operating hours) according to the Federal Airworthiness Requirement /Joint Aviation Requirement [FAR/JAR-25] has been added. [FAR/JAR-25] uses the general principle of an inverse relationship between the probability of a failure condition and the degree of hazard to the aircraft or its occupants. As [DO-178B] considers qualitative demonstration of software compliance to such high reliability to be beyond the current software technology, the [FAR/JAR-25] numbers are provided for information only.

Level A: Catastrophic failure

Failure conditions which would prevent continued safe flight and landing.
[FAR/JAR-25] extremely improbable, catastrophic failure < $1x10^{-9}$
These failure conditions are so unlikely that they are not anticipated to occur during the entire life of all aircraft of one type.

Level B: Hazardous/Severe-Major failure

Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:

- a large reduction in safety margins or functional capabilities;
- physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely;
- adverse effect on occupants including serious or potentially fatal injuries to a small number of those occupants.
  [FAR/JAR-25] extremely remote, $1x10^{-9}$ < hazardous failure < $1x10^{-7}$

Level C: Major failure

Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example:

- a significant reduction in safety margins or functional capabilities;
- a significant increase in crew workload or in conditions impairing crew efficiency, or
- discomfort to occupants, possibly including injuries.
  [FAR/JAR-25] remote, $1x10^{-7}$ < major failure < $1x10^{-5}$

Level D: Minor failure

Failure conditions which would not significantly reduce aircraft safety and which would involve crew actions that are well within their capabilities. Minor failure conditions may include for example:

- a slight reduction in safety margins or functional capabilities;
- a slight increase in crew workload, such as, routine flight plan changes or some inconvenience to occupants.
  [FAR/JAR-25] probable, minor failure > $1x10^{-5}$

Level E: No Effect

Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

3.3 Software life cycle

[DO-178B] deliberately refrains from making statements about appropriate software life cycle models. The life cycle is described rather abstract as a number of processes that are categorised as follows:

- software planning process which entails the production of the following documents:
  - plan for software aspects of certification. The main purpose of this document is to define the compliance of the chosen software development process to [DO-178B] for the certification authorities. This document contains many references to the project documentation generated as part of the applied life cycle model;
  - software development plan, which defines the chosen software life cycle and the software development environment, including all tools used;
  - software verification plan, which defines the means by which the verification objectives will be met;
  - software configuration management plan;
  - software quality assurance plan.
- software development processes consisting of :
  - software requirement process;
  - software design process;
  - software coding process;
  - integration process.
- integral processes which are divided into :
  - software verification process;
  - software configuration management process;
  - software quality assurance process;
  - certification liaison process.

The integral processes are a result of the criticality of the software. Consequently the integral processes are performed concurrently with the software development processes throughout the entire software life cycle.

3.4 Verification

In order to provide the developer with maximum flexibility, [DO-178B] allows the developer to choose the software life cycle model. [DO-178B] enforces traceability to its general requirements by verifying that the life cycle process provides all data it requires. Each constituent software development process has to be traceable, verifiable and consistent. Transition criteria need to be defined by the developer to determine whether the next software development process may be started. In case of iterative processes, like in the spiral model, attention needs to be paid to the verification of process inputs which become available after the subsequent process is started.

Verification is defined in [DO-178B] as "the evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards to that process". Review, analysis, test or any combination of these three activities can accomplish verification. Review provides a qualitative assessment of correctness.

Analysis is a detailed examination of a software component. It is a repeatable process that can be supported by tools. Every tool needs to be verified against the Tool Operational Requirements, the contents of which is prescribed in [DO-178B]. For software tools the same documentation and configuration control procedures apply as for the airborne software. Every software tool needs approval of the certification authority.
Testing is "the process of exercising a system or system components to verify that it satisfies specified requirements and to detect errors". By definition the actual testing of deliverable software forms only part of the verification of the coding and integration processes. For software classified at [DO-178B] level A, a mandatory 100% code coverage applies. This code coverage consists of :

- statement coverage (every statement executed, called statement testing in [BS7925-2]);
- decision coverage (every decision executed for pass and fail, called branch/decision testing in [BS7925-2]), and
- the modified condition/ decision coverage (mc/dc, same name in [BS7925-2]). Mc/dc requires that for every condition in a decision, its effect on the outcome of the decision is demonstrated.

Code coverage will be shown at module level testing.

4 Software development process

The definition of the software development process has been guided by previous experience with safety-critical software for spacecraft. More information on the spacecraft application is provided in [Dekk, Kess].

The project team was set up consisting of 2 separate groups, a development group and a verification group. The verification group was headed by a team member with sufficient authority to report, at his own discretion, to the company management outside the project hierarchy, in compliance with [DO-178B]. Furthermore the quality assurance manager was independent from both teams and not allowed to produce deliverable code or tests. The quality assurance manager needed his technical background in order to judge technical choices made.

The embedded application project started using :

- the DOD-STD-2167A life cycle model [DOD], which is based on the waterfall model ;
- customer supplied requirement specifications in plain English ;
- formal reviews after each life cycle phase;
- software analysis using Structured Analysis with Hatley and Pirbhai Real Time extensions (SA/RT) [Hatl, Pirb] supported by a Computer Aided Software Engineering (CASE) tool;
- software design using Yourdon Structured Design (SD) supported by the same CASE tool;
- the customer prescribed C language;
- NLR's proprietary C coding standard, with project specific enhancements and enforced by a static analysis tool;
- execution of module tests and integration tests on the target system;
- an automated test tool to aid the construction and cost effective repetition of the functional tests and code coverage tests;
- a proprietary configuration management tool;
- module and integration testing on the target with a simulated environment;
- integration with the aircraft avionics suite after integration of the embedded application.

5 Experience

5.1 DO-178B experience

Modern aircraft contain huge amounts of software, supplied by numerous independent suppliers world-wide. Even a single aircraft contains software of many different suppliers. According to the US National Transport Safety Board (NTSB), [DO-178B] works well as up to now no catastrophic failure (i.e. fatalities or hull losses) can be directly attributed to a software failure [IEEE]. An independent software engineering experiment using a [DO-178B] compliant software development process by NASA confirms that no errors were identified in the developed software [Hayh]. [DO-178B] contains sufficient information for first time users to implement a compliant software process.

5.2 Software classification

In the embedded application, software classified at levels A, B and E has been realised. Partitioning software is produced to allow software of various levels to run on the same processor. At the end of the project 81% of the modules are classified at level A, 8% at level B and 11% at level E. The increasing number of data fusion requirements lead to a larger share of level A software at the expense of level B software. With the small amount of level B modules remaining it is unclear whether the advantages of less rigorous testing of level B software outweigh the more complicated software development process.

When software classified at different levels has to run on the same processor, special partitioning software guarantees that software of one level can under no circumstance compromise the functioning of software at other levels. This partitioning software consumed only 1% of the total project effort. Even if all level B software would be developed as level A software, the partitioning software remains necessary and cost effective for separation of level A and level E (mainly maintenance) software.

5.3 C language

The C programming language contains numerous constructs that are unspecified, undefined or left to be defined by the compiler supplier [Hatt]. The C programming language is considered a project risk. This risk was reduced by choosing an ISO C-90 (also known as ANSI-C) compliant compiler complemented by a project coding standard defining, amongst others, a safe subset of C. Compliance to this project coding standard can be verified automatically by customising a commercial tool. The tool verification required by [DO-178B] revealed that the version management by the tool supplier turned out to be inadequate. The tool was already marketed world-wide since 1986 to hundreds of customers. This illustrates the rigour of the applied verification processes.

5.4 Requirements evolution

Due to the commercially defined short time-to-market, the customer defined the system requirements concurrently with the software requirement process. Before the start of the software design process the resulting analysis was subjected to a number of informal detailed technical assessments, performing the formal requirements verification activities with the exception of the certification authority involvement.

To aid the integration of the embedded application with the displays, co-developed by the customer, and subsequently with the avionics suite of the aircraft, a first version of the software with limited functionality was delivered before completion of the software requirements and software design processes. The first version served its purpose well. A lot of feed-back was obtained, resulting in many changes to and clarifications of the system requirements. Figure 1 depicts the resulting requirements evolution from the project start. Every point indicates a formal delivery of a working prototype or system to the customer. Figure 1 is cumulative: the number of partially implemented requirements is added to the number of fully implemented requirements. Superimposed is the number of requirement changes for each delivery. The status of a requirement in a delivery can be:

- fully implemented;
- partially implemented i.e. the delivery only partially complies with the requirement and additional work is needed arrive at full compliance;
- not implemented, i.e. no part of the requirements is included in the delivery.

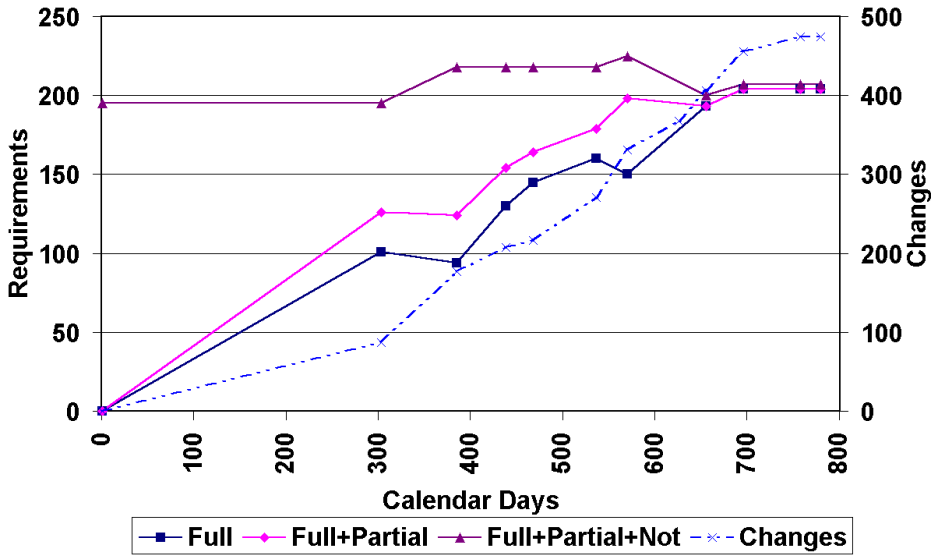

Fig. 1 Evolution of requirements and their implementation status

The increase in the number of requirements and the reduction in the number of implemented requirements after 300 and 520 working days are caused by new issues of the requirements document.
The changes are caused by (in descending order):

- changes in the Human Machine Interfaces (HMI) of the aircraft. These changes originate from pilot comments and can only be obtained from demonstrating a working prototype in a realistic environment. Co-development of the displays and the embedded application helps to reduce the amount of changes on system level;
- adding product features. Apart from marketing input, these changes also result from experience with an actual prototype;
- integration of the embedded application with the displays and the aircraft subsystems. Formal methods to specify these interfaces might have helped to reduce this class of changes;
- ambiguities in the plain English specifications. Especially for HMI related features an unambiguous specification method which is intelligible for pilots, HMI experts and computer software experts is needed.

The requirements evolution combined with the need for intermediate versions resulted in a change from the waterfall model to the spiral model. For the non-certified versions the formal reviews were replaced by technical reviews with the same contents but without the external attendants. The multiple deliveries implied frequent integration with the avionics suite at the customer's site. This resulted in the combination of our team with the customer's display team on one site. Of the 15 deliveries only the ones at 655 and 779 calendar days have been certified. Note that the non-certified versions are not to be used in flying aircraft.

5.5 Design evolution

Figure 2 shows the evolution of the number of modules (files containing C code) and external functions over time.

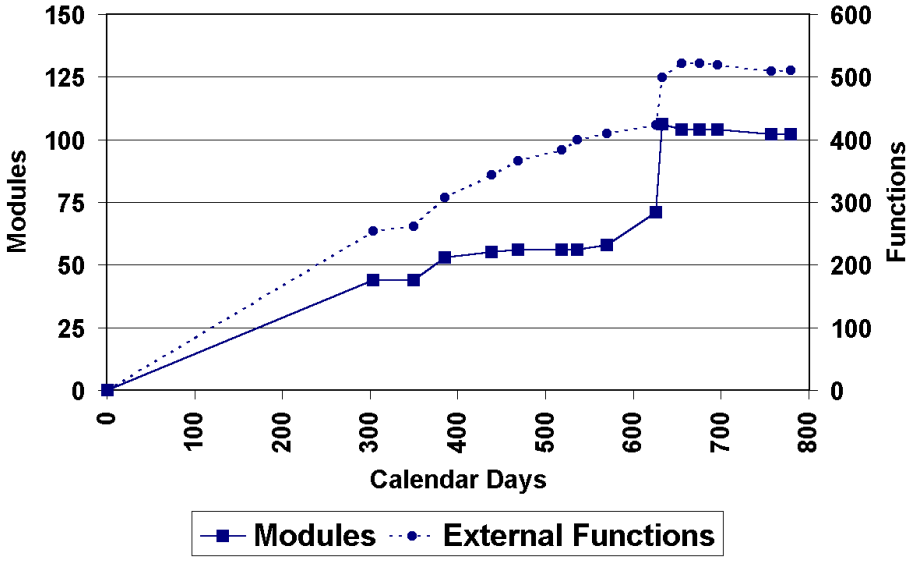

Fig. 2 Module evolution

Up until the first certified version the number of modules increased only slightly, indicating that all changes could be accommodated in the original design. Due to different verification requirements, software of different levels was split into different modules for certified versions. This splitting causes the sharp rise in the number of commonly developed modules just before the first certified version. Evolving data fusion requirements influenced the safety classification of some functions. Some simplifications of a communication protocol for the second certified version resulted in a minor reduction in the number of modules.

The number of external functions rose approximately continuously until the first certified version, in accordance with the number of implemented requirements. The number of functions remained virtually constant for the second certified version. This indicates that the design remained relatively stable, most requirement changes could be accommodated in the existing modules.

On average there are 5 functions per module. On average each file has been submitted to configuration control 13 times. These changes are concentrated in one configuration item, the second configuration item became stable after the version of day 536. The remaining 2 configuration items remained unchanged after the version of day 438.

These results support the view that also in an environment with significant requirement evolution a sufficiently mature design is needed before starting the coding process. The design team leader ensured that the design remained uncompromising during the entire realisation period.

5.6 Code size evolution

The code size evolution is shown in figure 3. Its continuous increase until the first certified version corresponds with the continuous increase in the number of implemented requirements. The subsequent slight reduction mirrors some requirements simplification.

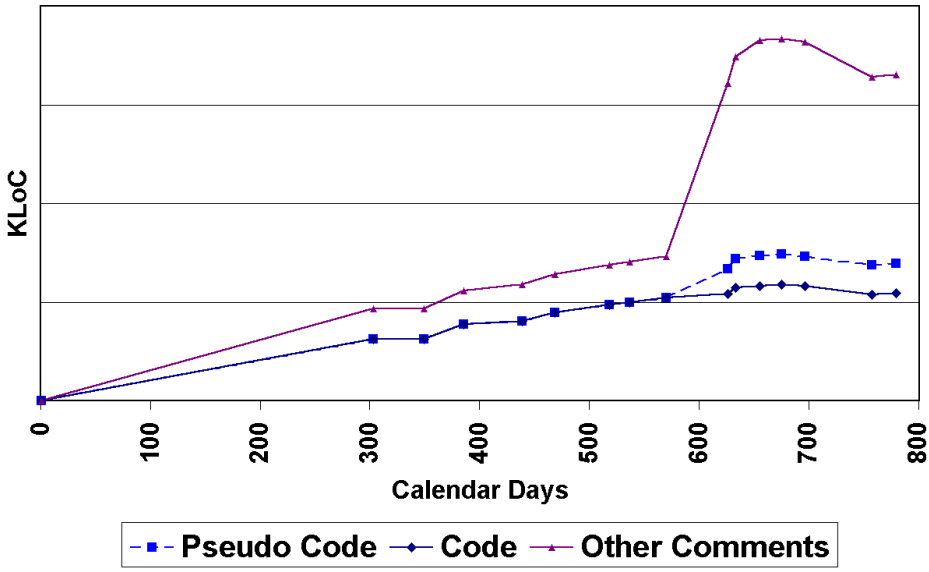

Fig. 3 Code size evolution

The CASE tool used only allows to progress once from analysis to design and once from design to code. It does not provide adequate support to incorporate analysis or design updates into the next phases. The amount of effort needed for data input even makes updating the analysis or design model cumbersome. After day 500 it was decided to retain the analysis model but limit its depth in order to allow for its updating.

The design model was abandoned as the CASE tool data input effort became unaffordable with the requirements evolution. Instead pseudo code was added to the code. The pseudo code contains an abstract description of the code in about 27% of its size. Also all interface definition information was added in extensive headers per function. This extra information explains the considerable increase in the amount of comment before the first certified version. The comment has a size of about 175% of the executable code.

On average each line of executable code has been modified 13.4 times, each comment line only 4.1 times. Changing the design information from the CASE tool to comment resulted in considerable man-hour savings, at the expense of a transition period with a less intelligible design. The design team leader and the verification team leader had sufficient knowledge to answer any question on the spot. With a maximum team size of 16 people located on one site this turned out to be a workable solution. The changes break down in about 60% changed lines, 15% deleted lines and 25% added lines. As the product grew in size over time more lines were added then deleted.

5.7 Code breakdown

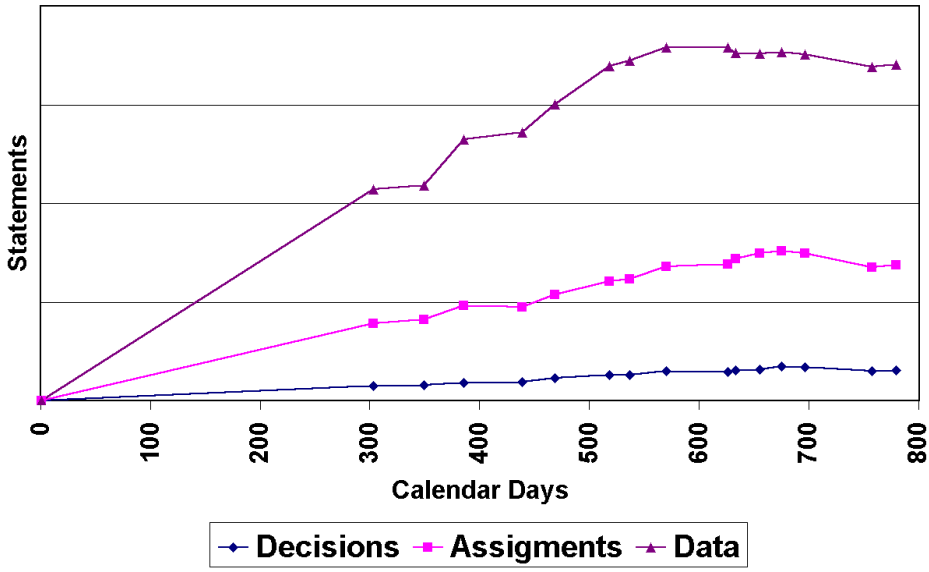

Fig. 4 Evolution of statement type distribution

For testing purposes a division of statements is made into :

- decisions and loops (consisting of the "switch", "if", "for" and "while" statements);
- assignments;
- data e.g. tables.

The results are shown in figure 4. All statement types increase approximately continuously until the first certified version, with a slight decrease up till the second certified version. The system design was already based on maximum configuration possibilities using data files. Adapting the software behaviour to specific aircraft configurations by configuration files has the advantage of obviating re-certification. The real-time constraints caused some run-time optimised solutions. Experience with the various prototypes lead to more sophisticated solutions

which comply with both the real-time requirements as well as with the requirements evolution. In the second certified version for each executable statement there is 1.48 lines of data. The statement type distribution reflects the design based on maximum use of data for configuring the software behaviour. The run-time optimisations are not reflected in a change of the statement type distribution.

5.8 Verification

Each testable requirement is identified to allow traceability from requirements through all development phases to verification. Every [DO-178B] compliant development phase contained full traceability of each requirement, by including the requirement identification. This has greatly helped the management of the virtually continuous requirement evolution. A lesson learned is to allocate a separate identification to each verifiable part of a requirement. [Hayh 1998] reached this conclusion independently.

A standard applies for the software requirement process. Its application has to be verified. Some simple tools can be produced to cost-effectively reduce the analysis effort. The same holds for the design standard.

For module tests the use of a Commercial-Of-The-Shelf (COTS) test tool greatly reduced the time needed to prepare the tests and to perform the regressions tests for each delivery. The actual test code is generated from Test Case Definition (TCD) files. On average each safety-critical function (i.e. [DO-178B] level A+B) is called 3.8 times during the verification tests. The non-comment part of the test case definition files equals 2.9 times the non-comment size of the code. The test comment grew to about 70% of the executable test case size implying that tool-assisted module testing still consumes a significant mount of effort. Due to the size of the test case definition files, comment is needed to document their function, to aid traceability, to improve readability, etc.

[DO-178B] requires data to be verified by inspection, only decisions and assignments can be verified by testing. For each testable statement 20 checks have been performed. For global data the test tool automatically checks that no global data is inadvertently changed, causing the large amount of checks per testable statement.

Integration testing was based on the white box approach. It comprised the correct functioning of combinations of functions. Integration tests also verified 19% of the requirements. These requirements could not be verified by black box testing only. Examples of the latter are spare processor time and spare memory requirements. During integration 184 tests have been performed. The COTS test tool did not support the multiple-module integration testing.

During validation testing the requirements are verified using a black box approach. Several requirements can be verified in one test. The 132 tests verified 90% of the requirements.

Analysis was used to verify 12% of the requirements. Note that some requirements can only be verified by a combination of analysis, validation testing and integration testing. Consequently the 3 percentages add up to more then 100%.

6 Conclusions

[DO-178B] compliant software processes have proven adequate for safety-critical software development.

Measuring the requirements evolution (refer figure 1) combined with the co-development need for intermediate versions resulted in the change from a waterfall software development process to a spiral software development process.

For a certifiable, safety-critical product with a commercially determined time-to-market co-development is a solution. The various prototypes, with increasing number of implemented requirements (refer figure 1), provided by a spiral software development process support this.

A sufficiently mature design is needed before starting the coding process for the first prototype. The design team leader has to ensure that the subsequent software modifications do not compromise the design. The module evolution (refer figure 2) needs to be complemented by other information to assess the design stability.

Metrics help in analysing and controlling the software processes. For example the evolution of requirements with their implementation status (refer figure 1) and the module evolution (refer figure 2), help in the trade-off between the date of the next delivery and its functions.

The CASE tool used did not adequately support design updates rendering it incompatible with the spiral model. Detailed design and interfaces can be included as comment in the code, to be automatically retrieved for the required documentation. The added source code (refer figure 3) turned out to be acceptable.

The statement type distribution (refer figure 4) reflects the maximum use of data to configure the software for each specific aircraft.

C combined with an appropriate coding standard and an automated analysis tool can be used for safety-critical certifiable software.

For some analysis tasks simple tools can be produced which cost-effectively reduce the analysis effort. The COTS test tool significantly reduced the testing effort.

References

[BS7925-2] British Standard software testing part 2: software components testing

(August 1998)

[Dekk, Kess] Product Assurance For The Development Of The SAX AOCS Application Software, G.J. Dekker, E. Kesseler (1996) ESA SP-377, NLR TP-96167

[DO-178B] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, (December 1992)

[DOD] DOD-STD-2167A Military Standard Defense System Software Development (1988)

[FAR/JAR-25] Federal Airworthiness Requirement/Joint Aviation Requirement FAR/JAR-25

[Hatl, Pirb] Strategies for real-time system specification, Hatley, D.J., Pirbhai, A. (1988) Dorset House Publishing

[Hatt] Safer C, Hatton L., (1995) Mc Graw-Hill

[Hayh] Framework for small-scale experiments software engineering, K. J. Hayhurst

[IEC 61508] IEC 61508 Functional safety: safety related systems, 7 parts, (June 1995)

[IEEE, 1998] IEEE, Developing software for safety- critical systems, J.Besnard, M. DeWalt, J. Voas, S. Keene (1998)

[ISO/DIS 15026] ISO/DIS 15026 Information technology - System and software integrity levels (1996)

[Kess, Slui] Safety and commercial realities in an avionics application, E. Kesseler, E. van de Sluis, Second World Congress on safety of transportation, NLR TP 97669 (1998)

[SAE ARP 4754] Society of Automotive Engineers Aerospace Recommended practise 4754, Certification considerations for highly-integrated or complex aircraft systems, (November 1996)